BigDawsSSH

Kenta Yoshii
kenta_yoshii@brown.edu

ABSTRACT

In BigDawsSSH, I have implemented two central protocols in the Secure Shell Protocol. Namely, the **Transport protocol** that focuses on establishing a secure and confidential connection between a server and a client and the **Authentication protocol** which then focuses on authenticating the user. **Connection Protocol**, which builds on top of these two protocols, is not implemented, as I deemed irrelevant to the class material.

1 INTRODUCTION

Secure Shell(SSH) is a network protocol that allows users to access and manage remote computers and servers over an unsecured network. Most notably, SSH provides a secure channel for data communication between two computers, encrypting any data that pass between them. In this project, I deep dived into the actual protocol to understand how this secure channel is established and how the server and the client are authenticated. In section 2, I will briefly go over some basic concepts in cryptography necessary to understand the workings of these protocols and explain how they are used. In section 3 of the report, I will explain in depth the two protocols needed for the secure channel to be established. In section 4, I will explain BigDawsSSH more in detail. In the final section, I will discuss some difficulties I encountered along the way.

2 PREREQUISITE KNOWLEDGE

Diffie-Hellman Key Exchange. Diffie-Hellman key exchange is an algorithm used to derive a shared secret between two parties over a public channel. Since this alogrithm was discussed in-depth in class already, I will not explain the way this shared secret is derived. In the context of SSH, DH key exchange is used to derive a shared secret **k**. This k then is also used to derive a unique hash **H** which uniquely identifies the SSH session. In Diffie-Hellman, the group used determines the strength of the keys used in the exchange. In SSH and BigDawsSSH, the following two groups are supported

$$diffie - hellman - group1 - sha1$$

$$diffie - hellman - group14 - sha1$$

Which of these two is used is determined in the algorithm negotation stage which takes place in the Transport protocol. In SSH, Key Re-exchange is also supported. If that happens, a new instance of DH key exchange is run, k' is computed, and keys for encryption and integrity are renewed. However, the unique hash H remains unchanged.

Symmetric Key Encryption. Symmetrical encryption is a type of encryption where one key can be used to encrypt messages to the opposite party, and also to decrypt the messages received from the other participant. Symmetric keys are used by SSH in order to encrypt the entire connection. These keys are session-based and can be renewed if either party wishes so. The actual derivation of this symmetric key and starting of data encryption take place at the end of key exchange and will be discussed in more detail in section 3 Transport protocol. SSH offers many different kinds of ciphers, but in BigDawsSSH the mainly used three are selected.

$$3des - cbc$$

$$aes128 - cbc$$

$$aes256 - cbc$$

Until the key exchange is done, no data encryption is done.

Asymmetric Key Encryption. Asymmetrical encryption is different from symmetrical encryption as two associated keys are needed. The first key, public key, is used to encrypt messages and nothing else can be achieved using it (it only goes in one-way). The second key, private key, is the only component capable of decrypting messages that were encrypted using the associated public key. By virtue of this fact, any entity capable of decrypting these messages has demonstrated that they are in control of the private key. SSH takes advantage of this fact and use asymmetric encryption to perform authentication. In BigDawsSSH, there are two such places.

(1) Server Host Authentication
The client starts up knowing the SSH server's public key. They then encrypt all the messages they send using the server's public key until shared secret is established. This way, the client can be sure that they are indeed talking to the server. For this part in BigDawsSSH,
$$ssh - dss$$
is used.
(2) Client Authentication
This takes place in the second protocol, Authentication protocol. The actual process is discussed in section 3. For this part in BigDawsSSH,
$$ssh - rsa$$

is used.

Hashing. Cryptographic hash functions are methods of creating a succinct "signature" or summary of a set of information. They are very difficult to be reversed (e.g., guessing x from $h(x)$ is difficult) and practically unique. Using the same hashing function and message should produce the same hash; modifying any portion of the data should produce an entirely different hash. Hence, a client/server should not be able to produce the original message from a given hash, but they should be able to tell if a given message produced a given hash. Given these properties, the main use of hashing in SSH is with HMAC, or hash-based message authentication codes. These are used to ensure the message text that's received is intact and unmodified. The specific hashing schemes supported in BigDawsSSH are

$$hmac - sha1$$

$$hmac - sha96$$

Similarly to encryption scheme, hmac scheme is also negotiated with the client in the Transport protocol. After the key derivation, HMAC are computed on unencrypted packet and attached to the encrypted packet for the receiver to verify. More on this later.

3 SSH PROTOCOLS

3.1 Transport Protocol

When a client first connects to the server, the client and the server will perform a protocol called **Transport protocol**. This protocol is used for the two parties to reach an agreement on the following important things.

(1) **SSH Version**
This describes the SSH version, software version and other various features for that session. Each party will exchange a protocol version upon connection.
(2) **Algorithms**
These include encryption algorithm, MAC scheme, compression algorithm, and key exchange algorithm among other things.
(3) **Shared Secret**
This shared secret is derived from the key exchange algorithm agreed above. It is used to derive symmetric keys for the other algorithms for later use
(4) **Exchange Hash (Session Identifier)**
This hash is also used in tandem with shared secret to derive cryptographic keys. It also serves as an unique session identifier, which will be used in the authentication protocol.
(5) **Cryptographic Keys**
These keys are the final output from this protocol. They are used to encrypt/decrypt and sign/verify any future data that will travel between the two to maintain confidentiality and integrity.

Server Authentication. In the proposal, we briefly discussed about the potential Man in the Middle attack that could happen. The RFC does not strictly enforce a server authentication but in BigDawsSSH, this is enforced. Each SSH client will have a local store of public keys of different SSH servers it may talk to. After the protocol version exchange and up until encryption of packets begin, the SSH server will sign the packets it send out using its corresponding secret key. The clients will perform a verification on each of these packet to make sure that they are indeed coming from the server. In BigDawsSSH, ssh-dss is used for this purpose.

Algorithm Negotiation. Upon completing the version exchange, algorithm negotation immediately begins. Each side will send the other party name-lists of algorithms (in the order of preference). A certain set of rules (e.g., the server also needs to support the algorithm the client prefers the most) are used to determine the algorithms that will be used in the later protocol. In this stage of the transport protocol, the following things are agreed upon

(1) **Key Exchange Algorithm** that takes place in the next stage to get a shared secret
(2) **Server Host Key Algorithm**
(3) **Encryption Scheme** that will be used for the duration of this session
(4) **MAC Algorithm** to preserve message integrity
(5) **Compression Algorithm** if any
(6) **Language** to be used in the particular session

Key Exchange. Among many other things, both sides will agree on a key exchange algorithm. In BigDawsSSH, both diffie-hellman-group1-sha1 and diffie-hellman-group14-sha1 are supported. The steps are follows:

(1) Client randomly samples a and computes g^a. Client then sends its public value $e = g^a$ to the server
(2) Server randomly samples b and computes $f = g^b$. It also computes $k = e^b$. It then computes the hash as

$$H = hash(V_C||V_S||I_C||I_S||K_S||e||f||k)$$

where V is the raw protocol version message bits (so V_C represents client's protocol version message bits), I is the raw negotiation message bits, and K_S is the server's host public key. The server then signs this hash using its host private key and send to client

$$(K_S, f, signature)$$

(3) Client computes k and H. It then verifies the signature on the hash.
(4) After the key exchange is complete, each side will have an exchange hash, H, and shared secret k.

Key Generation. After the key exchange, each side will generate 6 different values, which are

(1) Client to Server Encryption Key
(2) Server to Client Encryption Key
(3) Client to Server Integrity Key
(4) Server to Client Integrity Key
(5) Client to Server Initial IV
(6) Server to Client Initial IV

In order for both sides to have the exact same keys, these keys are generated in a deterministic way. For example, Client to Server Initial IV is computed by

$$hash(k||H||"A"||session\_id)$$

Other keys are computed likewise. Server to Client Initial IV is computed by substituting "A" with "B".

Here it is important that the keys and IVs are sufficient in size and in accordance with the agreed upon algorithms. For example, if the agreed upon encryption algorithm is $aes256 - cbc$, the key needs to be 256 bits in length and IVs 16 bits (or the block size of cbc) in length.

The reason why we generate a new key for each direction(e.g., client $\rightarrow$ server) is to prevent mirror attack (e.g., relaying packets sent out by server back to the server) from happening.

Binary Packet Protocol. Binary Paclet Protocol or BPP is the underlying protocol used to transfer data between the client and the server. In order to prevent from malicious party conducting a detail analysis on each packet, a random padding is added to the actual payload. Then *packet_length*, *padding_length*, *payload*, and *random_padding* are all encrypted with the agreed algorithm using the derived key. HMAC is also computed on the raw unencrypted Binary Packet to provide message integrity (Encrypt-and-MAC). Via these two measures, SSH makes sure the connection is both secure and confidential.

3.2 Authentication Protocol

After a secure and confidential channel is established between the two parties, Authentication protocol is run to authenticate the SSH client. This protocol is initiated by the client issuing a service request at the end of the transport protocol (ssh-auth). In BigDawsSSH, two types of authentication schemes, further discussed below, are supported. In the beginning, the server sends the list of available authetication methods to the client. Client can choose any method in any order.

3.2.1 Publickey. In this scheme, a user authenticates itself by proving that they possess a secret signing key. An user will generate a key-pair in advance and send register the public key on the server. Then when the actual authentication protocol is run, the user can simply provide its username, public key, and signature computed over some other metadata and signed by the corresponding secret key. If the server successfully verifies the signature using the registered public key, then the client is considered authenticated.

In BigDawsSSH, ssh-rsa is used to perform a publickey based user authentication.

3.2.2 Password. In this scheme, a user is prompted a password. If the provided password exists in the SSH server's database, then that client is considered authenticated. We note that since every Binary Packet is encrypted and HMACed, password is just sent as a plaintext (no salt and/or pepper needed).

After a user is successfully authenticated, then they can start using the services (e.g., ssh-connection) that are offered by the server host.

4 BIGDAWSSSH

In this section, I will discuss a little more in detail how BigDawsSSH is implemented.

4.1 Setup

When the server binary starts, the following things are loaded

(1) Server's name-lists for algorithm negotiation
(2) Server's DSA keys for server host authentication
(3) Registered users' passwords for clients' password authenticating (if they choose so)

When the client binary starts, the following things are loaded

(1) Client's name-lists for algorithm negotiation
(2) Server's DSA public key for authenticating the server

4.2 Pre-Key Exchange

In this part, no encryption or HMAC is performed. Hence in the Binary Packet, MAC field is left as empty.

Protocol Version Exchange is performed following the rules described in RFC 4253.

Algorithm Negotiation is also performed exactly following the steps described in the RFC. To focus on cryptographic parts of this protocol, I decided to simply ignore Compression and Language, although shared values for both of them are reached.

4.3 Key Exchange/Keys Generations

BigDawsSSH follows the Key Exchange/Keys Generation steps exactly. It however does not support Key Re-exchange yet.

4.4 Client Authentication

When the Transport protocol is done, the user is prompted for the following inputs

(1) Username
(2) (Authentication) Method
(3) Password (if *password* scheme is used)

Depending on the input for method, a different protocol is run with the server. Eventually, the user succeeds in authenticating itself and the protocol will be over.

5 DISCUSSION

Here I list some difficulties I encountered along the way and how I solved them

(1) Understanding the Protocol itself
    Arguably the most difficult part was understanding the protocol itself. For this I simply read the relevant RFCs over and over until I was able to understand all the different aspects of the protocol.
(2) Handling packets
    Another difficult part for me was dealing with various different kinds of messages, from negotiatio message to key exchange message, and fitting all of them into the format Binary Packet Protocol was expecting. To solve this, I used variosu struct definitions for each and every kind of messages and defning their own marshall/unmarshall functions.
(3) Support for more than one schemes
    Since SSH should be able to support more than one encryption/HMAC/Authentication schemes, I also struggled making BigDawsSSH do the same. Initial design was not conductive into doing this because everything was kind of hardcoded. To add support for more than one schemes, I first made everything very generic. For example, I added a handler function that invokes a specific function associated with that particular scheme.

6 REFERENCE

(1) RFC4250 (Message Code, etc.)
(2) RFC4251 (Protocol architecture)
(3) RFC4252 (Authentication protocol)
(4) RFC4253 (Transport protocol)